PHP Extension Writing

Sara Golemon

Yahoo! Inc
search.yahoo.com
pollita@php.net

May 15, 2007

Hyatt Regency Chicago, Illinois

- Why are you here?
  - Glue in some external lib?
  - Speed up some stable business logic?
  - Distribute a closed-source library?
  - Have a little fun?
- How fast should I go?
  - Keep it simple?
  - Warp speed?
  - Ludicrous speed?

- Setup a build environment:
  - Build from source
  - Install your distro's -dev/_devel package
  - VMWare Workstation / ext|tek image
- Don't forget build tools
  - autoconf-2.13
  - automake-1.4+
  - libtool-1.4.x (except 1.4.2)
  - bison-1.28, 1.35, 1.75, 2.0+
  - flex-2.5.4
  - re2c-0.9.11+

Rough Schedule

- Part One: Introduction to PHP:
  - Lifecycles
  - Memory Management
  - Variables
- Part Two: Simple Extensions
  - Constants
  - Exporting functions
  - Working with arrays
  - Extension Globals
  - INI Settings
- Part Three: Objects
  - PHP4 compatable ones
  - Visibility scoped PHP5 style

- Lifecycles:
  - STARTUP, ACTIVATION, RUNTIME, DEACTIVATION, SHUTDOWN
- Memory Management:
  - Persistent / Non-Persistent
- Userspace Accessible Variables:
  - Labels, Values, and References

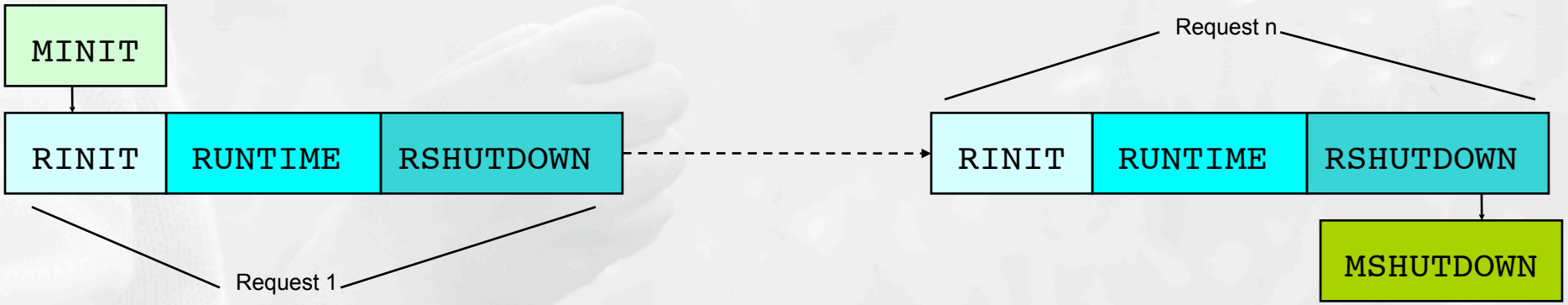
- STARTUP
  - Initial startup of a PHP process space (either by command line invocation or Apache child)
  - Initialize engine and core components
  - Parse php.ini
  - Initialize (MINIT) staticly built modules
  - Initialize (MINIT) shared modules loaded by php.ini
  - Finalize Initialization

- ACTIVATION
  - Triggered upon receiving a new request (page hit).
  - Initialize environment and variables (symbol_table, EGPCS)
  - Activate (RINIT) staticly built modules
  - Activate (RINIT) shared modules

- RUNTIME
  - Actual execution of scripts happens here.
  - Compile and execute auto_prepend_file.
  - Compile and execute main_file.
  - Compile and execute auto_append_file.
  - During execution, script interacts with extension via:
    - » Function Calls
    - » Class Instantiation
    - » Indirect APIs (INI Set, Streams, PDO, etc...)

- DEACTIVATION
  - Upon exit(), die(), E_ERROR, or end of script execution.
  - Call user-defined shutdown functions.
  - Destroy object instances.
  - Flush output.
  - Deactivate (RSHUTDOWN) modules (in reverse of activation order).
  - Clean up environment
  - Implicitly free any remaining non-persistent allocations.

- SHUTDOWN
  - Final good-night. Called as process space is terminating (apache child termination).
  - Shutdown (MSHUTDOWN) all modules (rev. startup order).
  - Shutdown the engine.



- Persistent Memory
  - Fancy name for normal memory allocation.
  - Sticks around (between requests if need-be) until explicitly freed by the application/extension.
  - Doesn't count towards mempory_limit
- Non-Persistent Memory
  - May only be allocated during a request (RINIT/RUNTIME/RSHUTDOWN)
  - Automatically freed by the engine if you don’t do it explicitly (last part of Deactivation stage)
  - Often can't be shared with 3rd party libs

```
void *emalloc(size_t count);
void *ecalloc(size_t nmemb, size_t size);
void *erealloc(void *ptr, size_t size);
void *estrdup(const char *ptr);
void *estrndup(const char *ptr, size_t size);
void *safe_emalloc(size_t nmemb,
                                   size_t size,size_t count);
void *STR_EMPTY_ALLOC(void);
void efree(void *ptr);
```

```
void *pemalloc(size_t count, int persist);
void *pecalloc(size_t nmemb, size_t size, int persist);
void *perealloc(void *ptr, size_t size, int persist);
void *pestrdup(const char *ptr, int persist);
void *zend_strndup(const char *ptr, unsigned int len);
void *safe_pemalloc(size_t nmemb,size_t size,size_t
    count, int persist);
void pefree(void *ptr, int persist);
```

- safe_pemalloc() does not exist prior to PHP 5.1.0

Output

- `stdout` is unsuitable for PHP since the controlling web server (especially threaded ones) may need the content directed to a specific pipe or buffer.
- Sending output to `stdout` would also bypass output buffering mechanisms such as zlib compression.
- To output data using the same mechanisms as a userspace `echo`/`print` statement, use one of:
  - `php_printf(const char *format, ...)`
  - `PHPWRITE(const char *str, int strlen)`
  - `zend_print_variable(zval *pzv);`

```
php_error(int type, const char *format, ...);
php_error_docref(const char *docref TSRMLS_DC,
                 int type, const char *foramt, ...);
```

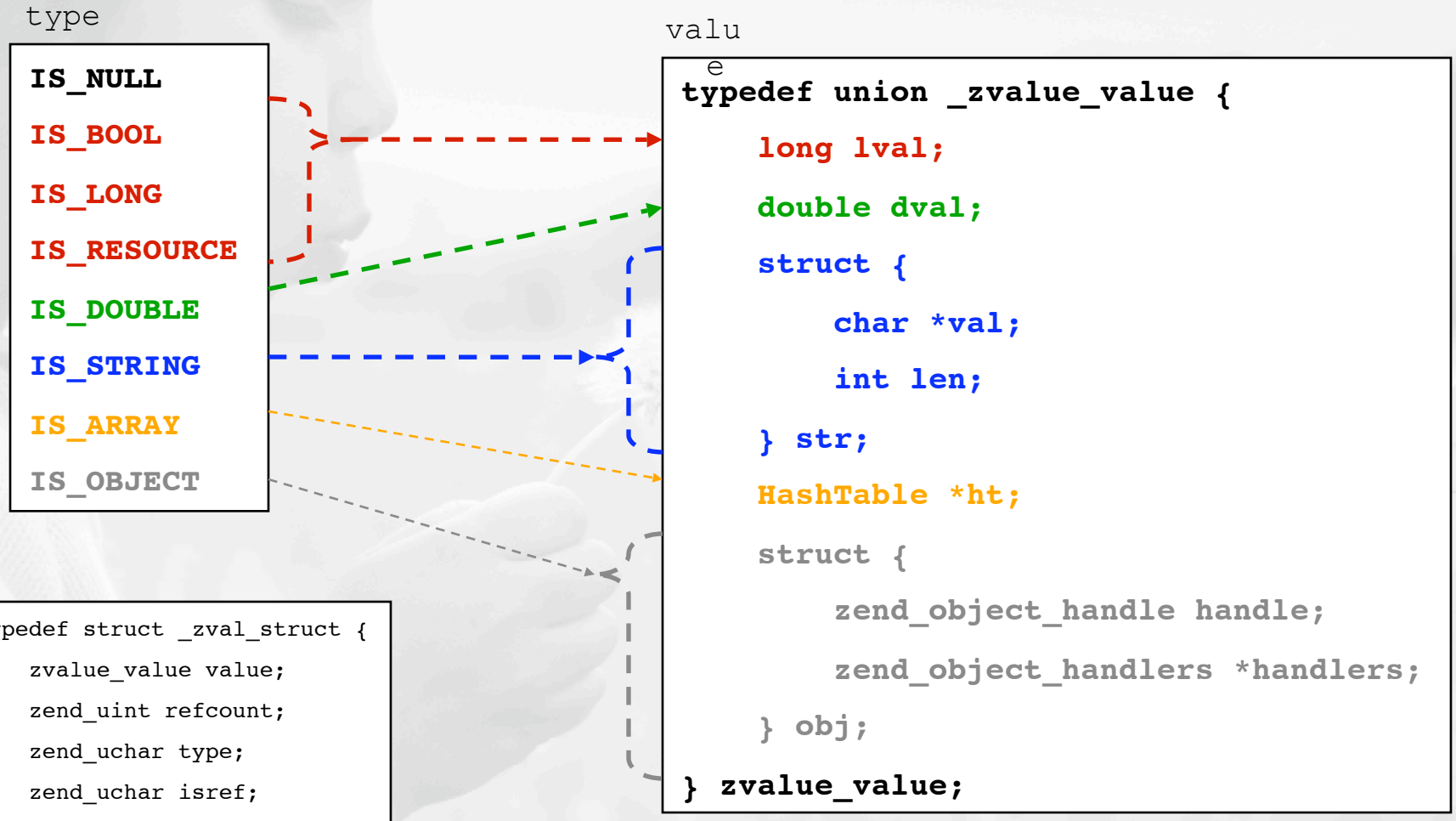
- docref: e.g. "`ref.errorfunc`"
  - » Used for generating docref error messages
  - » `NULL` means "current function"
- type: `E_WARNING` et. al. - See http://php.net/ref.errorfunc
  - » `E_ERROR` ends the request

- A userspace variable ($foo) is made up of two parts:
  - Label: The name of the variable (e.g. foo)
  - Value: Actual contents of the variable:
    - NULL
    - Bool(TRUE)
    - Int(123)
    - Float(3.1415926535)
    - String(3) “bar”
    - Resource#1(Stream)
    - Object#2(stdClass) {
      [bar] => baz
      }
    - Array(1) {
      [bar] => baz
      }

Userspave variables values

- Variable's value: `zval` (aka: `pval`)
  - `type`: One of the primitive datatypes
  - `value`: Actual value stored in a union
  - `isref`: Style of reference:1 or 0
    - » 0: Copy-on-write Reference
    - » 1: Full-reference
  - `refcount`: Labels referencing this `zval`

zval: Type and Value




```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar isref;
} zval;
```

Accessing zvals

- Z_* macros come in tripplet sets.
  - Z_XXX(zv) – Operates on an immediate zval
  - Z_XXX_P(pzv) – Operates on zval*
  - Z_XXX_PP(ppzv) – Operates on a zval**

Z_TYPE(zv)	zv.type
Z_BVAL(zv)	zv.value.lval
Z_LVAL(zv)	zv.value.lval
Z_DVAL(zv)	zv.value.dval
Z_STRVAL(zv)	zv.value.str.val
Z_STRLEN(zv)	zv.value.str.len
Z_RESVAL(zv)	zv.value.lval
Z_ARRVAL(zv)	zv.value.ht

PHP4

Z_OBJCE(zv)	zv.value.obj.ce
Z_OBJPROP(zv)	zv.value.obj.properties

PHP5+

Z_OBJVAL(zv)	zv.value.obj
Z_OBJ_HANDLE(zv)	Z_OBJVAL(zv).handle
Z_OBJ_HT(zv)	Z_OBJVAL(zv).handlers
Z_OBJ_HANDLER(zv, hf)	Z_OBJ_HT((zv))->hf
Z_OBJCE(zv)	zend_get_class_entry(&(zv) TSRMLS_CC)
Z_OBJPROP(zv)	Z_OBJ_HT((zv))-> get_properties(&(zv) TSRMLS_CC)

- Two definitions of the word “reference”
  - Userspace: http://php.net/language.references
  - Internals: All variables are references
    - Copy-on-write reference set: $a = 123; $b = $a;
    - Full-Reference (Userspace defn): $a = 123; $b = &$a;
- In both cases, labels share the same zval* value.

  pzv->type == IS_LONG;

  pzv->value.lval == 123;

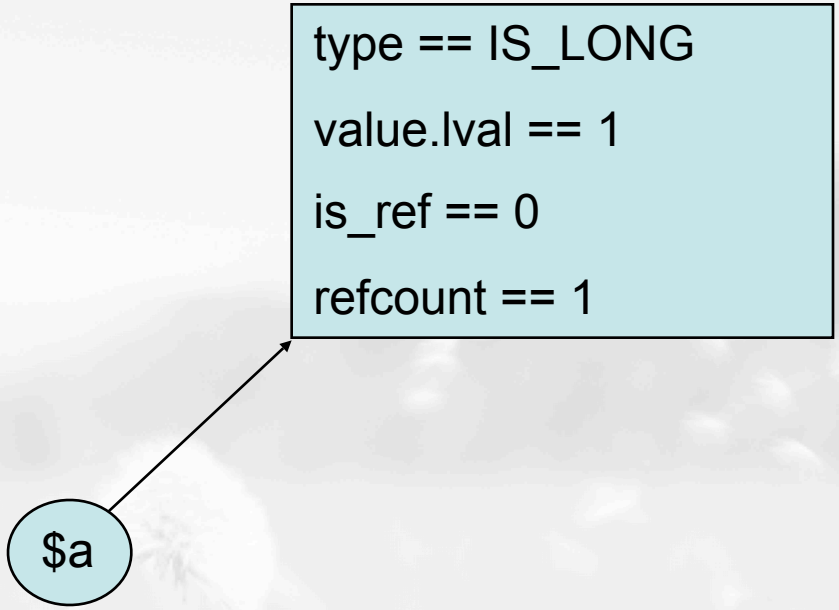
  pzv->refcount == 2

  pzv->is_ref == 0 for Copy-on-write,

  1 for Full-reference

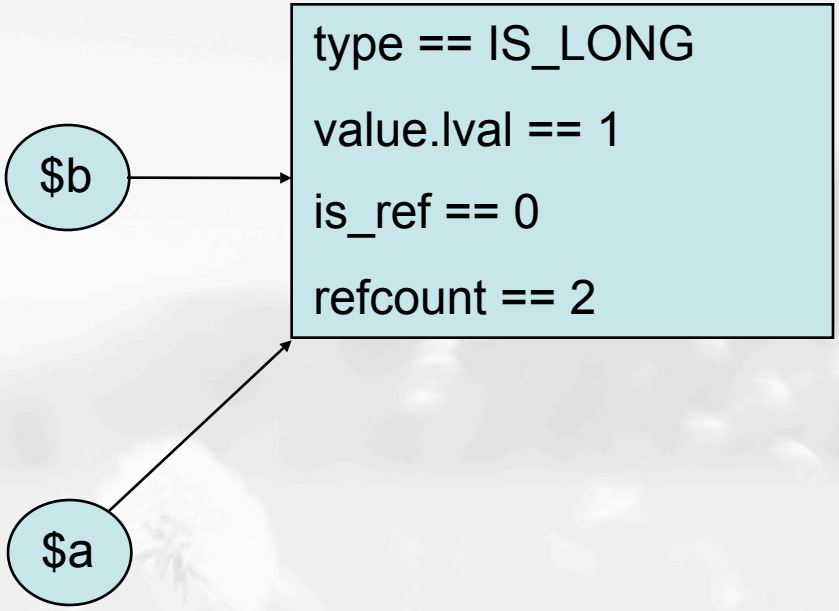
Reference Counting

```
$a = 1;
```



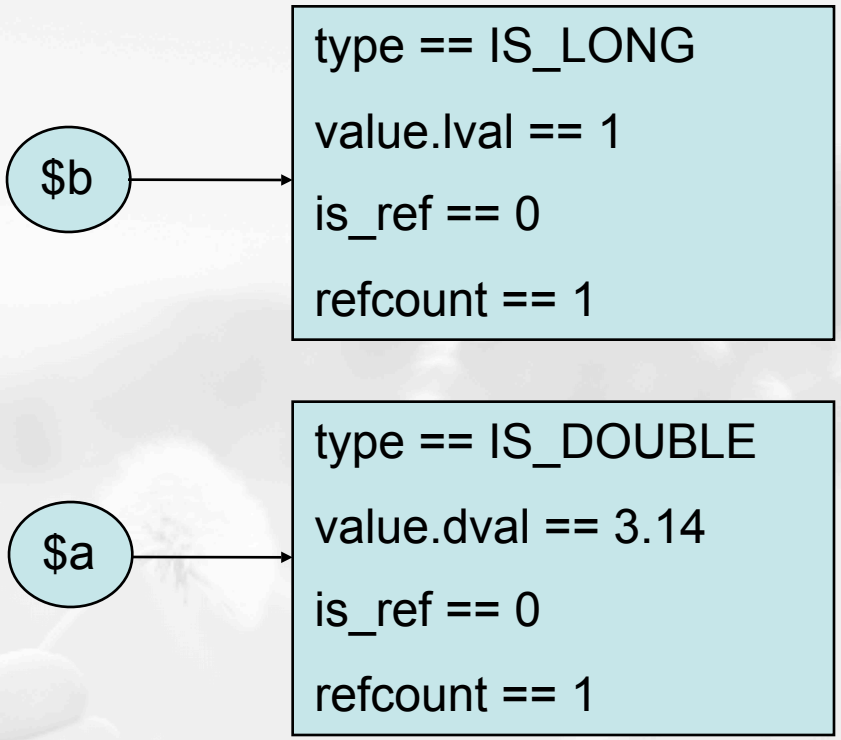
Reference Counting

```
$a = 1;
$b = $a;
```



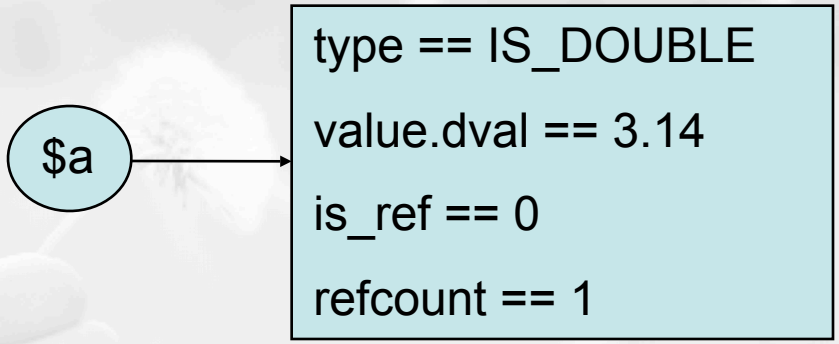
Reference Counting

```
$a = 1;
$b = $a;
$a = 3.14;
```



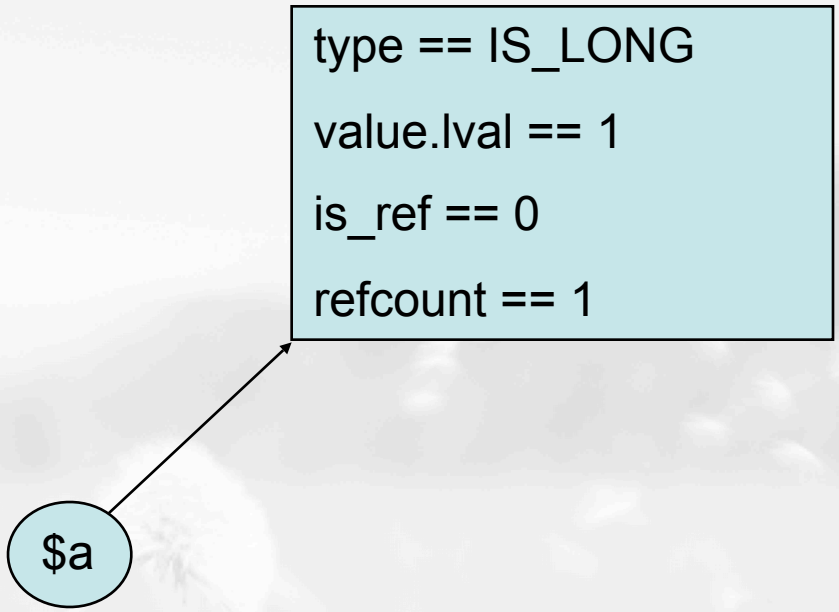
Reference Counting

```
$a = 1;
$b = $a;
$a = 3.14;
unset($b);
```



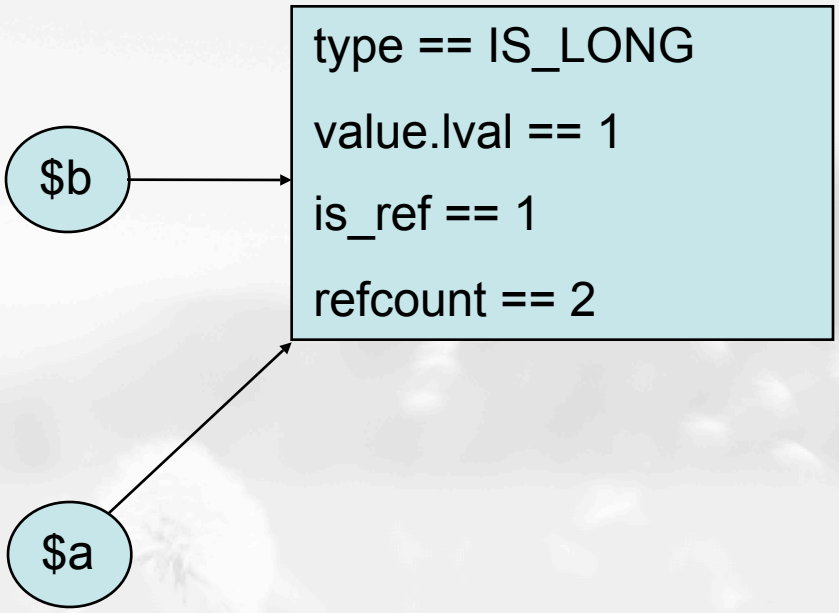
Reference Counting

```
$a = 1;
```



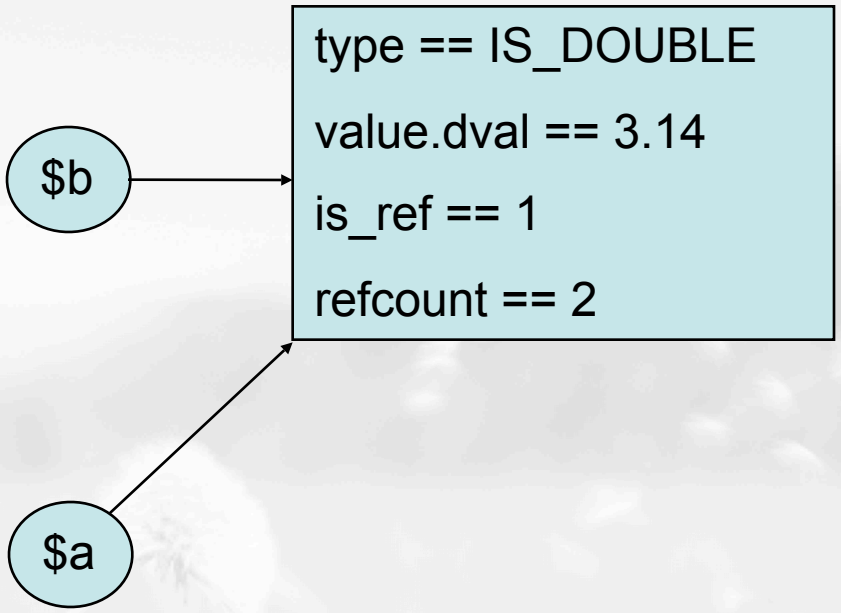
Reference Counting

```
$a = 1;
$b =& $a;
```



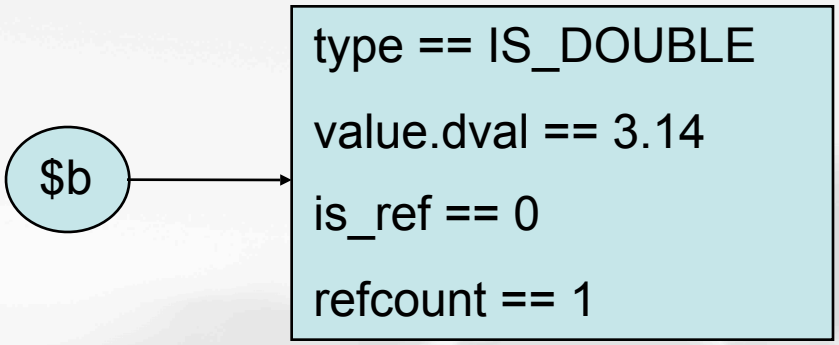
Reference Counting

```
$a = 1;
$b =& $a;
$a = 3.14;
```



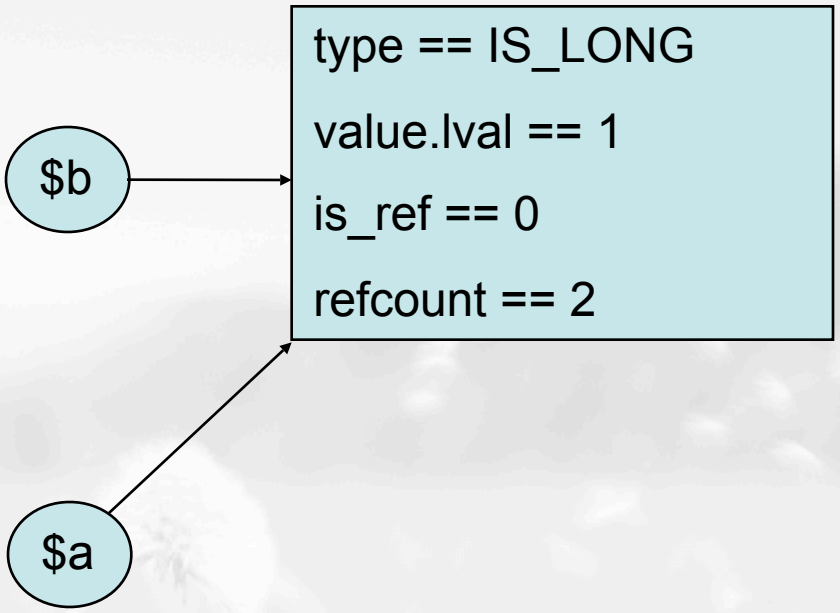
Reference Counting

```
$a = 1;
$b =& $a;
$a = 3.14;
unset($a);
```



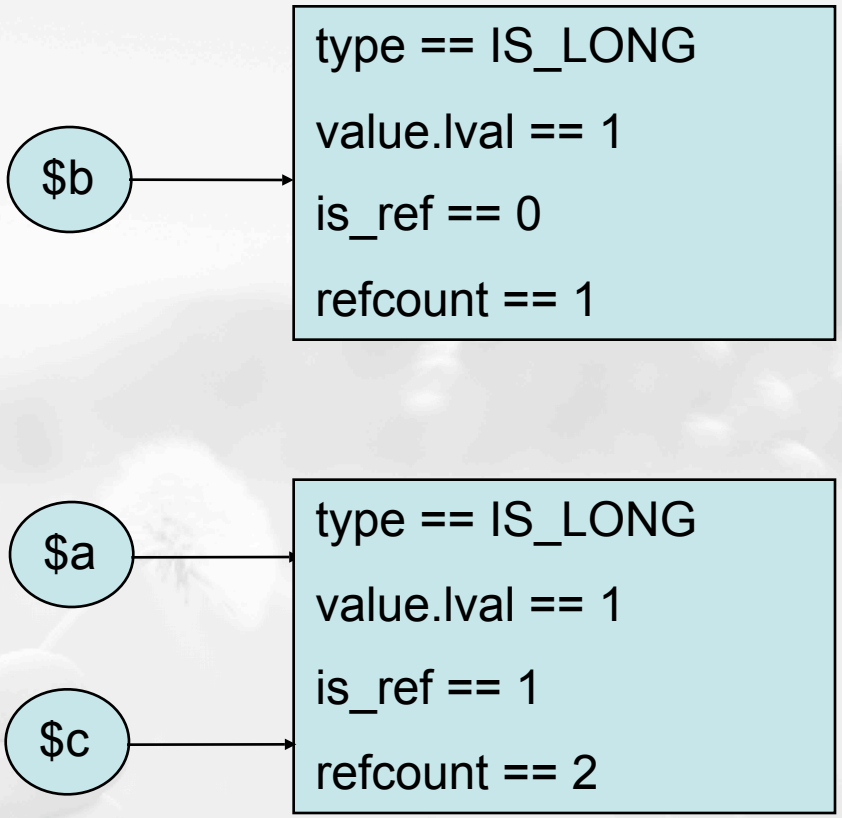
Reference Counting

```
$a = 1;
$b = $a;
```



Reference Counting

```
$a = 1;
$b = $a;
$c =& $a;
```



- How are we doing so far?
- Got that build environment setup?
- Need to stretch?

- Constants
- Functions
- Arrays
- Extension Globals
- INI Settings

- config.m4 helps build ./configure script
- Syntax is simple for simple extensions, convoluted for complex extensions

```
PHP_ARG_ENABLE(myext, whether to enable myExt,
[  --enable-myext          Enable My first extension])

if test "$PHP_MYEXT" != "no"; then
 PHP_NEW_EXTENSION(myext, php_myext.c, $ext_shared)
fi
```

config.m4

- `zend_module_entry` introduces your ext
- Exports initial process hooks

```
zend_module_entry myext_module_entry = {
        STANDARD_MODULE_HEADER,
        "myext",
        NULL, /* functions */
        NULL, /* MINIT */
        NULL, /* MSHUTDOWN */
        NULL, /* RINIT */
        NULL, /* RSHUTDOWN */
        NULL, /* MINFO */
        NO_VERSION_YET,
        STANDARD_MODULE_PROPERTIES
};
```

php_myext.c

- Static compilation
  - /usr/local/src/php-5.3.0 $ ./configure --enable-myext

```
extern zend_module_entry myext_module_entry;
#define phpext_myext_ptr &myext_module_entry
```

php_myext.h

- Dynamic compilation
  - ../code/myext/01 $ phpize && ./configure

```
#ifdef COMPILE_DL_MYEXT
ZEND_GET_MODULE(myext)
#endif
```

php_myext.c

  - C++/PHP4 needs `extern "C" { }` declaration

Ready to build

- `phpize`
  - Uses `config.m4` and already installed PHP headers to generate `./configure` script
- `./configure`
  - Plumbs local machine environment to generate `Makefile`s and local configuration values
- `make all`
  - Builds the extension
- `make install`
  - Installs shared object to `extension_dir` location
  - You need to add `extension=myext.so` yourself

- Make sure it loads
  - php -m
  - php --re myext
- Most common causes why it won't:
  - `PHP Warning: PHP Startup: Invalid library (maybe not a PHP library) 'myext.so' in Unknown on line 0`
    - Missing `ZEND_GET_MODULE()`
    - Using C++ with PHP4 and didn't extern `ZEND_GET_MODULE()`

```
extern "C" { ZEND_GET_MODULE(myext) }
```

- Initialized during STARTUP phase
  - Easy to do LONG, DOUBLE, and STRING types
  - Annoying to make BOOL type
  - Not recommended to make others
  - Macros expect string literal for name

```
PHP_MINIT_FUNCTION(myext)
{
    REGISTER_LONG_CONSTANT   ("MYEXT_MEANING", 42,
                                        CONST_CS | CONST_PERSISTENT);
    REGISTER_STRING_CONSTANT("MYEXT_NAME",     "Nombre",
                                        CONST_CS | CONST_PERSISTENT);

    return SUCCESS;
}
```

- Initialized during ACTIVATION phase
  - Same rules apply
  - Don't use CONST_PERSISTENT
  - Watch out for non-persistent string constants!
  - "Okay" to register Resources here
    - » STDIN, STDOUT, STDERR

```
PHP_RINIT_FUNCTION(myext)
{
    REGISTER_DOUBLE_CONSTANT("MYEXT_VERSION", 3.14, CONST_CS);

    REGISTER_STRINGL_CONSTANT("MYEXT_NOMBRE", estrdup("Sara"),
                                sizeof("Sara") - 1, CONST_CS);

    return SUCCESS;
}
```

- Only register callbacks you need
- Ignore MINFO for now

```
zend_module_entry myext_module_entry = {
        STANDARD_MODULE_HEADER,
        "myext",
        NULL, /* functions */
        PHP_MINIT (myext),
        NULL, /* MSHUTDOWN */
        PHP_RINIT (myext),
        NULL, /* RSHUTDOWN */
        NULL, /* MINFO */
        NO_VERSION_YET,
        STANDARD_MODULE_PROPERTIES
};
```

Boolean (and other) Constants

- Annoying, isn't it?

```
PHP_MINIT_FUNCTION(simple)
{
    zend_constant c;

    c.value.type = IS_BOOL;
    c.value.value.lval = 1;
    c.flags = CONST_CS | CONST_PERSISTENT;
    c.name_len = sizeof("MYEXT_TRUTH");
    c.name = zend_strndup("MYEXT_TRUTH", c.name_len - 1);
    c.module_number = module_number;
    zend_register_constant(&c TSRMLS_CC);

    return SUCCESS;
```

- Declare using `PHP_FUNCTION()` macro
- Can generate output with `php_printf()`

```
PHP_FUNCTION(myext_hello_world)
{
    php_printf("Hello World!\n");
}
```

- Bind functions using `zend_function_entry`

```
zend_function_entry php_myext_functions[] = {
    PHP_FE(myext_hello_world,       NULL)
    { NULL, NULL, NULL }
};
```

- Module points to function list
- Function list points to functions

```
zend_module_entry myext_module_entry = {
        STANDARD_MODULE_HEADER,
        "simple",
        php_myext_functions,
        PHP_MINIT(myext),
        NULL, /* MSHUTDOWN */
        PHP_RINIT(myext),
        NULL, /* RSHUTDOWN */
        NULL, /* MINFO */
        NO_VERSION_YET,
        STANDARD_MODULE_PROPERTIES
};
```

- Additional functions added with more FEs
- Always keep the NULL trio at the end

```
zend_function_entry php_myext_functions[] = {
    PHP_FE(myext_hello_world,       NULL)
    PHP_FE(myext_greeting,          NULL)
    { NULL, NULL, NULL }
};
```

- `zend_parse_parameters()`
  - Converts userspace variables to C types

```
PHP_FUNCTION(myext_greeting)
{
    char *name;
    int name_len;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "s", &name, &name_len) == FAILURE) {
        return;
    }
    php_printf("Hello %s, nice to meet you!\n", name);
}
```

zend_parse_parameters() format specifiers

b	zend_bool	Boolean
l	long	Long (integer)
d	double	Double (float)
s	char *, int	String (value and length)
r	zval *	Resource
a	zval *	Array
o	zval *	Object (of any type)
z	zval *	Any type
Z	zval **	Any type

|	All further arguments are optional
!	Do not overwrite current variable contents if a userspace NULL is passed.
/	Automatically separate the variable passed (force a copy)

More fun with arguments

```
PHP_FUNCTION(myext_math) {
    double a, b;
    zend_bool divide = 0;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                                        "dd|b", &a, &b, &divide) == FAILURE) {
        return;
    }
    if (divide) {
        php_printf("%f / %f == %f\n", a, b, a / b);
    } else {
        php_printf("%f * %f == %f\n", a, b, a * b);
    }
}
```

Returning arguments


```
PHP_FUNCTION(myext_math2) {
    double a, b;
    zend_bool add = 1;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                                        "dd|b", &a, &b, &add) == FAILURE) {
        return;
    }
    if (add) {
        RETURN_DOUBLE(a + b);
    } else {
        RETURN_DOUBLE(a - b);
    }
}
```

Other value returning macros

RETURN_NULL();	NULL
RETURN_TRUE;	bool(true)
RETURN_FALSE;	bool(false)
RETURN_BOOL(b);	True for any non-zero integer value
RETURN_LONG(l);	Integer of specified value
RETURN_DOUBLE(d);	Float of specified value
RETURN_STRING(s,dup);	NULL terminated string, duplicated if necessary
RETURN_STRINGL(s,l,dup);	String of specified length, optionally duplicated
RETURN_EMPTY_STRING()	Just like the name says...
RETURN_RESOURCE(r)	Registered resource instance

- Anything you return to userspace has to be safe for the engine to efree() later on.
- (dup)licate any non-emalloc'd strings

More on returning values

RETURN_NULL();	RETVAL_NULL(); return;
RETURN_BOOL(b);	RETVAL_BOOL(b); return;
RETURN_TRUE;	RETVAL_TRUE; return;
RETURN_FALSE;	RETVAL_FALSE; return;
RETURN_LONG(l);	RETVAL_LONG(l); return;
RETURN_DOUBLE(d);	RETVAL_DOUBLE(d); return;
RETURN_STRING(s, dup);	RETVAL_STRING(s, dup); return;
RETURN_STRINGL(s, len, dup);	RETVAL_STRINGL(s, len, dup); return;
RETURN_RESOURCE(r);	RETVAL_RESOURCE(r); return;

- Set a value, then `return;`

Yet more on return values...

RETVAL_NULL();	ZVAL_NULL(return_value);
RETVAL_BOOL(b);	ZVAL_BOOL(return_value, b);
RETVAL_TRUE;	ZVAL_TRUE(return_value);
RETVAL_FALSE;	ZVAL_FALSE(return_value);
RETVAL_LONG(l);	ZVAL_LONG(return_value, l);
RETVAL_DOUBLE(d);	ZVAL_DOUBLE(return_value, d);
RETVAL_STRING(s, dup);	ZVAL_STRING(return_value, s, dup);
RETVAL_STRINGL(s, len, dup);	ZVAL_STRINGL(return_value, s, len, dup);
RETVAL_RESOURCE(r);	ZVAL_RESOURCE(return_value, r);

- `zval *return_value`
  - Passed in through `PHP_FUNCTION()` declaration

What those ZVAL macros really do

ZVAL_NULL(pzv);	pzv->type = IS_NULL;
ZVAL_BOOL(pzv, b);	pzv->type = IS_BOOL; pzv->value.lval = b;
ZVAL_TRUE(pzv);	pzv->type = IS_BOOL; pzv->value.lval = 1;
ZVAL_FALSE(pzv);	pzv->type = IS_BOOL; pzv->value.lval = 0;
ZVAL_LONG(pzv, l);	pzv->type = IS_LONG; pzv->value.lval = l;
ZVAL_DOUBLE(pzv, d);	pzv->type = IS_DOUBLE; pzv->value.dval = d;

ZVAL_STRING(pzv, s, dup);	pzv->type = IS_STRING;
	pzv->value.str.val = dup ? estrdup(s) : s;
	pzv->value.str.len = strlen(s);

ZVAL_STRINGL(pzv, s, len, dup);	pzv->type = IS_STRING;
	pzv->value.str.val = dup ? estrndup(s,len) : s;
	pzv->value.str.len = len;

ZVAL_RESOURCE(pzv, r);	pzv->type = IS_RESOURCE; pzv->value.lval = r;

array_init(pzv);	$pzv = array();
add_next_index_long(pzv, 42);	$pzv[] = 42;
add_index_double(pzv, 10, 3.14);	$pzv[10] = 3.14;
add_assoc_string(pzv, "foo", "bar", 1);	$pzv['foo'] = 'bar';

- Internal arrays created like userspace arrays
- All the same scalar types are supported

- `range(0, 99);`

```
PHP_FUNCTION(myarray_range100) {
    int i;
    array_init(return_value);
    for(i = 0; i < 100; i++) {
        add_next_index_long(return_value, i);
    }
}
```

- Simple enough to be readable

```
PHP_FUNCTION(myarray_100range) {
    int i;
    array_init(return_value);
    for(i = 0; i < 100; i++) {
        zval *value;
        MAKE_STD_ZVAL(value);
        ZVAL_LONG(value, i);
        add_next_index_zval(return_value, value);
    }
}
```

- Functionally identical to previous example

```
PHP_FUNCTION(myarray_100range) {
    int i;
    array_init(return_value);
    for(i = 0; i < 100; i++) {
        zval *value;
        pzv = emalloc(sizeof(zval));       INIT_PZVAL(pzv);
        ZVAL_LONG(value, i);
        add_next_index_zval(return_value, value);
    }
}
```

- Functionally identical to previous example

What about subarrays?


```
PHP_FUNCTION(myarray_list) {
    zval *person;

    array_init(return_value);                                   /* $ret = array();         */
    MAKE_STD_ZVAL(person);    array_init(person); /* $p = array();          */
    add_assoc_string(person, "name", "Bob", 1);         /* $p['name'] = 'Bob';  */
    add_assoc_long(person, "age", 42);                  /* $p['age'] = 42;         */
    add_next_index_zval(return_value, person);          /* $ret[] = $p;            */
    MAKE_STD_ZVAL(person);    array_init(person); /* $p = array();          */
    add_assoc_string(person, "name", "Alice", 1);       /* $p['name'] = 'Alice'; */
    add_assoc_long(person, "age", 31);                  /* $p['age'] = 31;         */
    add_next_index_zval(return_value, person);          /* $ret[] = $p;            */
}
```

Remember refcounts?


```
PHP_FUNCTION(myarray_twobobs) {
    zval *person;

    array_init(return_value);                                   /* $ret = array();         */
    MAKE_STD_ZVAL(person);    array_init(person); /* $p = array();          */
    add_assoc_string(person, "name", "Bob", 1);         /* $p['name'] = 'Bob';  */
    add_assoc_long(person, "age", 42);                  /* $p['age'] = 42;         */
    add_next_index_zval(return_value, person);          /* $ret[] = $p;            */
    ZVAL_ADDREF(person);
    add_assoc_zval(return_value, "Bob", person);      /* $ret['Bob'] = $p;       */
}
```

How can I read an array?

```
PHP_FUNCTION(myarray_greeting) {
    zval *person, **name;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                                        "a", &person) == FAILURE) {
        return;
    }
    if (zend_hash_find(Z_ARRVAL_P(person), "name", sizeof("name"),
                        &name) == SUCCESS && Z_TYPE_PP(name) == IS_STRING) {
        php_printf("Hello %s\n", Z_STRVAL_PP(name));
    }
}
```

- "a" checks type automatically
- We need to check name's type manually

Other ways to examine an array

zend_hash_num_elements(pht)	count($arr)
zend_hash_exists(pht, key, keysize)	isset($arr[$key])
zend_hash_index_exists(pht, idx)	isset($arr[$idx])
zend_hash_find(pht, key, keysize, pppvz)	$val = $arr[$key]
zend_hash_index_find(pht, idx, pppvz)	$val = $arr[$idx]

- Exists calls return `1`(true) or `0`(false)
- Find calls return `SUCCESS` or `FAILURE`
- `keysize` *includes* terminating NULL
  - zend_hash_find(pht, "foo", 4, &ppzv)
- sizeof("literal") / strlen("literal") + 1
  - zend_hash_find(pht, "bar", sizeof("bar"), &ppzv)

```
PHP_FUNCTION(myarray_sum) {
    zval *arr, **curr;    int total = 0;
    HashTable *pht;     HashPosition pos;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,  "a", &arr) == FAILURE)
                                                                { return; }
    pht = Z_ARRVAL_P(arr);
    for(zend_hash_internal_pointer_reset_ex(pht, &pos);
          zend_hash_get_current_data_ex(pht, &curr, &pos) == SUCCESS;
          zend_hash_move_forward_ex(pht, &pos)) {
        if (Z_TYPE_PP(curr) == IS_LONG)  {   total += Z_LVAL_PP(curr);   }
    }
    RETURN_LONG(total);
}
```

- HashPosition points at current element

Other ways to examine an array

zend_hash_internal_pointer_reset_ex(pht, ppos)	reset($arr)
zend_hash_move_forward_ex(pht, ppos)	next($arr)
zend_hash_move_backward_ex(pht, ppos)	prev($arr)
zend_hash_internal_pointer_end_ex(pht, ppos)	end($arr)
zend_hash_get_pointer(pht, ppos)	
zend_hash_set_pointer(pht, ppos)	
zend_hash_has_more_elements_ex(pht, ppos)	key($arr) !== false

- Passing NULL for ppos uses "Internal" ptr
- Most have direct userspace analogues
  - get/set_pointer() wouldn't make sense
- has_more returns `SUCCESS/FAILURE`

zend_hash_get_current_key_ex (pht, pstr, plen, pidx, dup, ppos)	key($arr)
zend_hash_get_current_data_ex(pht, pppzv, ppos)	current($arr)
zend_hash_get_current_key_type_ex(pht, ppos)	is_long(key($arr)) is_string(key($arr))
zend_hash_update_current_key_ex(pht, type, str, len, idx, ppos)	

- get_current_key*() funcs return:
  - `HASH_KEY_IS_LONG`
  - `HASH_KEY_IS_STRING`
  - `HASH_KEY_NON_EXISTANT`
- Others return `SUCCES / FAILURE`

Callback based itteration: array_walk()

zend_hash_apply(pht, pcallback TSRML_CC)	array_walk($arr, 'myfunc')
int noargs_callback(zval **data TSRMLS_DC)	myfunc($val)

zend_hash_apply_with_argument (pht, pcallback, parg TSRMLS_CC)	array_walk ($arr, 'myfunc', $arg)
int onearg_callback (zval **data, void *arg TSRMLS_DC)	myfunc($val, $dummy, $arg)

zend_hash_apply_with_arguments (pht, pcallback, count, ...)	array_walk ($arr, 'myfunc', array(...))
int multiarg_callback(zval **data, int count, va_list args, zend_hash_key *key)	myfunc($val, $key, $args)

ZEND_HASH_APPLY_REMOVE	foreach($a as $k => $v) { /* ... */ unset($a[$k]); continue; }
ZEND_HASH_APPLY_STOP	foreach($a as $k => $v) { /*...*/ break; }

Itterating Arrays - "Move Forward" Method


```
int sum_func(zval **data, int *total TSRMLS_DC) {
    if (Z_TYPE_PP(data) == IS_LONG) {
        *total += Z_LVAL_PP(data);
    }
    return ZEND_HASH_APPLY_KEEP;
}

PHP_FUNCTION(myarray_sum2) {
    zval *arr;    int total = 0;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,  "a", &arr) == FAILURE)
                                                                { return; }
    zend_hash_apply_with_argument(Z_ARRVAL_P(arr), sum_func, &total TSRMLS_CC);
    RETURN_LONG(total);
}
```

Good time for that question
you were holding onto...

I know I blew through that section

- Remember, PHP might be threading...
- Making actual globals becomes a bad idea

```
long number = 0;
PHP_FUNCTION(badidea_setnum) {
    long num;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &num) == FAILURE) {
        return;
    }
    number = num;
}
PHP_FUNCTION(badidea_getnum) {
    RETURN_LONG(number);
}
```

BAD

IDEA

Thread
Safe
Resource
Manager
Local
Storage

Extension Globals See: /code/myext/04/

php_myext.h

```
ZEND_BEGIN_MODULE_GLOBALS(myext)
    /* List your globals here */
    long number;
ZEND_END_MODULE_GLOBALS(myext)

/* This part is 99% copy/paste */
#ifdef ZTS
# include "TSRM.h"
# define MYEXT_G(v) TSRMG(myext_globals_id, \
                          zend_myext_globals *, v)
#else
# define MYEXT_G(v) (myext_globals.v)
```

Declare them in source


```
ZEND_DECLARE_MODULE_GLOBALS(myext);

void globals_ctor(zend_myext_globals *myext_globals TSRMLS_DC)
{ myext_globals->number = 0; }
void globals_dtor(zend_myext_globals *myext_globals TSRMLS_DC) { }

PHP_MINIT_FUNCTION(myext) {
    ZEND_INIT_MODULE_GLOBALS(myext, globals_ctor, globals_dtor);
    return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(myext) {
#ifdef ZTS
    ts_free_id(myext_globals_id);
#else
    globals_dtor(&myext_globals TSRMLS_CC);
#endif
```

Accessing data


```
PHP_FUNCTION(myext_setnum)
{
    long num;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &num)==FAILURE) {
        return;
    }
    MYEXT_G(number) = num;
}

PHP_FUNCTION(myext_getnum)
{
    RETURN_LONG(MYEXT_G(number));
}

PHP_RINIT_FUNCTION(myext)
{
    MYEXT_G(number) = 0;
```

- All INI entries are (initially) strings, even numbers

```
#include "php_ini.h"
PHP_INI_BEGIN()
  PHP_INI_ENTRY("myext.greeting", "Hello", PHP_INI_ALL, NULL)
  PHP_INI_ENTRY("myext.enabled",  "1",     PHP_INI_ALL, NULL)
PHP_INI_END()

PHP_MINIT_FUNCTION(myext)
{
    REGISTER_INI_ENTRIES();
    return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(myext)
{
    UNREGISTER_INI_ENTRIES();
    return SUCCESS;
```

INI_INT(char *setting)	INI_ORIG_INT(char *setting)
INI_FLT(char *setting)	INI_ORIG_FLT(char *setting)
INI_BOOL(char *setting)	INI_ORIG_BOOL(char *setting)
INI_STR(char *setting)	INI_ORIG_STR(char *setting)

- INI_XXX() gives current (request) setting
- INI_ORIG_XXX() gives global (system) setting
- String value re-converted on each call

Crontrolling when settings can be changed

PHP_INI_SYSTEM	php.ini or httpd.conf only
PHP_INI_PERDIR	httpd.conf <directory> or .htaccess
PHP_INI_USER	ini_set()
PHP_INI_ALL	PHP_INI_SYSTEM | PHP_INI_PERDIR | PHP_INI_USER

- `PHP_INI_PERDIR` usually *doesn't* stand alone
  - `PHP_INI_SYSTEM | PHP_INI_PERDIR`
- Modification callbacks can provider mroe control

Binding INI entries to Extension Globals


```
ZEND_BEGIN_MODULE_GLOBALS(myext)
    char *greeting;
    long type;
ZEND_END_MODULE_GLOBALS(myext)
```

```
PHP_INI_BEGIN()
  STD_PHP_INI_ENTRY("myext.greeting", "Hello", PHP_INI_ALL,
      OnUpdateString, greeting, zend_myext_globals, myext_globals)
  STD_PHP_INI_ENTRY("myext.type", "1", PHP_INI_ALL,
      OnUpdateLong,   type,     zend_myext_globals, myext_globals)
PHP_INI_END()

void globals_ctor(zend_myext_globals *myext_globals TSRMLS_DC)
{
    myext_globals->greeting = NULL;
}
```

OnUpdate Handlers

OnUpdateBool	zend_bool
OnUpdateLong	long
OnUpdateReal	double
OnUpdateString	char*
OnUpdateStringUnempty	char*
OnUpdateEncoding (PHP6)	UConverter*

- PHP4: OnUpdateInt (not Long)
- Custom OnUpdate methods allowed
  - OnUpdateSafeMode
  - OnUpdateBaseDir

- Objects
  - PHP4 compatable ones
  - Visibility scoped PHP5 style

Declaring empty classes See: /code/myobject/01


```
zend_class_entry *MyClass_ce;

PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyClass", NULL);
    MyClass_ce = zend_register_internal_class(&ce TSRMLS_CC);

    return SUCCESS;
}
```

- Creates an empty class definition
  - `class MyClass { }`

Inheriting


```
zend_class_entry *MyClass_ce, *MyChild_ce;

PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyClass", NULL);
    MyClass_ce = zend_register_internal_class(&ce TSRMLS_CC);
    INIT_CLASS_ENTRY(ce, "MyChild", NULL);
    MyChild_ce = zend_register_internal_class_ex(&ce,
                                MyClass_ce, "myclass" TSRMLS_CC);
    return SUCCESS;
}
```

- `class MyClass { }`
  - `class MyChild extends MyClass { }`
  - Specify parent `ce` **or** parent name

```
zend_class_entry *MyInt_ce, *MyAbs_ce;

PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyInterface", NULL);
    MyInt_ce = zend_register_internal_class(&ce TSRMLS_CC);
    MyInt_ce->ce_flags |= ZEND_ACC_INTERFACE;
    INIT_CLASS_ENTRY(ce, "MyAbstract", NULL);
    MyAbs_ce = zend_register_internal_class(&ce TSRMLS_CC);
    MyAbs_ce->ce_flags |= ZEND_ACC_EXPLICIT_ABSTRACT_CLASS;
    return SUCCESS;
}
```

- `interface MyInterface { }`
- `abstract classs MyAbstract { }`

Define class methods (PHP4 Compat)


```
PHP_FUNCTION(MyClass_methodOne)
{
    /* Do stuff */
}

zend_function_entry MyClass_methods[] = {
    PHP_NAMED_FE(methodone, PHP_FN(MyClass_methodOne), NULL)
    { NULL, NULL, NULL }
};
```

```
/* snip */
    INIT_CLASS_ENTRY(ce, "MyClass", MyClass_methods);
/* snip */
```

Define class methods (PHP5 or Higher)


```
PHP_METHOD(MyAbs, methodOne)
{
    /* Do stuff */
}

zend_function_entry MyAbs_methods[] = {
    PHP_ME(MyAbs, methodOne, NULL, ZEND_ACC_PUBLIC)
    PHP_ABSTRACT_ME(methodTwo, NULL, ZEND_ACC_PUBLIC)
    { NULL, NULL, NULL }
};
```

ZEND_ACC_PUBLIC	ZEND_ACC_CTOR
ZEND_ACC_PROTECTED	ZEND_ACC_DTOR
ZEND_ACC_PRIVATE	ZEND_ACC_STATIC
	ZEND_ACC_ABSTRACT
>= 5.2	ZEND_ACC_DEPRECATED

Define class methods (PHP5 or Higher)


```
ZEND_BEGIN_ARG_INFO(MyAbs_methodTwo_arginfo, 0)
  ZEND_ARG_INFO       (0, name)
  ZEND_ARG_ARRAY_INFO(0, options,              1)
  ZEND_ARG_OBJ_INFO   (0, db,       "PDODriver", 0)
  ZEND_ARG_INFO       (1, errorMsg)
ZEND_END_ARG_INFO()

zend_function_entry MyAbs_methods[] = {
    PHP_ME(MyAbs, methodOne, NULL, ZEND_ACC_PUBLIC)
    PHP_ABSTRACT_ME(MyAbs, methodTwo, MyAbs_methodTwo_arginfo)
    { NULL, NULL, NULL }
};
```

- `abstract public function($name, Array $options = NULL, PDODriver $db, &$errorMsg);`

Defining class constants (PHP5 only)


```
PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyClass", MyClass_methods);
    MyClass_ce = zend_register_internal_class(&ce TSRMLS_CC);
    zend_declare_class_constant_string(MyClass_ce, "VERSION",
        sizeof("VERSION") - 1, "1.2.3" TSRMLS_CC);
    return SUCCESS;
}
```

- `class MyClass { const VERSION = '1.2.3'; }`
- null, bool, long, double, string, stringl
- String constants must be persistent!

Defining default properties (PHP5 only)


```
PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyClass", MyClass_methods);
    MyClass_ce = zend_register_internal_class(&ce TSRMLS_CC);
    zend_declare_property_bool(MyClass_ce,
         "enabled", sizeof("enabled") - 1, 1,
         ZEND_ACC_PUBLIC TSRMLS_CC);
    return SUCCESS;
}
```

- `class MyClass { public $enabled = true; }`
- Same rules as constants

Defining static properties (PHP5 only)


```
PHP_MINIT_FUNCTION(myobject)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "MyClass", MyClass_methods);
    MyClass_ce = zend_register_internal_class(&ce TSRMLS_CC);
    zend_declare_property_null(MyClass_ce,
        "instance", sizeof("instance") - 1,
        ZEND_ACC_PUBLIC | ZEND_ACC_STATIC TSRMLS_CC);
    return SUCCESS;
}
```

- `class MyClass { public static $instance = null; }`

Implementing a Constructor (and then some)


```
PHP_METHOD(MyClass, __construct)
{
    char *a, *b;
    int a_len, b_len;
    if (zend_parse_parameters_ex(
                ZEND_PARSE_PARAMS_QUIET, ZEND_NUM_ARGS() TSRMLS_CC,
                "ss", &a, &a_len, &b, &b_len) == FAILURE) {
        zend_throw_exception(zend_get_error_exception(),
                                "Invalid arguments", -1 TSRMLS_CC);
        return;
    }
    zend_update_property_stringl(EG(scope), getThis(),
                    "valOne", sizeof("valOne") - 1,
                    a, a_len TSRMLS_CC);
    zend_update_property_stringl(EG(scope), getThis(),
                    "valTwo", sizeof("valTwo") - 1,
                    b, b_len TSRMLS_CC);
}
```

Implementing a Constructor (and then some)


```
PHP_METHOD(MyClass, __construct)
{
    char *a, *b;
    int a_len, b_len;
    if (zend_parse_parameters_ex(
                ZEND_PARSE_PARAMS_QUIET, ZEND_NUM_ARGS() TSRMLS_CC,
                "ss", &a, &a_len, &b, &b_len) == FAILURE) {
        zend_throw_exception(zend_get_error_exception(),
                                "Invalid arguments", -1 TSRMLS_CC);
        return;
    }
    zend_update_property_stringl(EG(scope), getThis(),
                    "valOne", sizeof("valOne") - 1,
                    a, a_len TSRMLS_CC);
    zend_update_property_stringl(EG(scope), getThis(),
                    "valTwo", sizeof("valTwo") - 1,
                    b, b_len TSRMLS_CC);
}
```

Places to go for more information

- Mailing Lists

  pecl-dev@lists.php.net (nntp|http)://news.php.net/php.pecl.dev

  internals@lists.php.net (nntp|http)://news.php.net/php.internals
- http://php.net/zend
- http://zend.com/php/internals
- "Extending and Embedding PHP"

  ISBN: 0-6723-2704-X
- IRC: eFnet/#php.pecl
- http://blog.libssh2.org
  - What the heck is TSRMLS anyway?
  - How long is a piece of string
  - You're being lied to...